Steven Englehardt*, Jeffrey Han, and Arvind Narayanan*

I never signed up for this! Privacy implications of email tracking

Abstract: We show that the simple act of viewing emails contains privacy pitfalls for the unwary. We assembled a corpus of commercial mailing-list emails, and find a network of hundreds of third parties that track email recipients via methods such as embedded pixels. About 30% of emails leak the recipient's email address to one or more of these third parties when they are viewed. In the majority of cases, these leaks are intentional on the part of email senders, and further leaks occur if the recipient clicks links in emails. Mail servers and clients may employ a variety of defenses, but we analyze 16 servers and clients and find that they are far from comprehensive. We propose, prototype, and evaluate a new defense, namely stripping tracking tags from emails based on enhanced versions of existing web tracking protection lists.

1 Introduction

Email began as a non-interactive protocol for sending simple textual messages. But modern email clients support much of the functionality of the web, and the explosion of third-party web tracking has also extended to emails, especially mailing lists. Surprisingly, while there is a vast literature on web tracking, email tracking has seen little research.

The ostensible purpose of email tracking is for senders to know which emails have been read by which recipients. Numerous companies offer such services to email senders [11, 14, 22], and mail clients that have privacy features advertise them as a way for users to protect their privacy from email *senders* [20, 31, 42]. But we find that email tracking is far more sophisticated: a large network of third parties also receive this information, and it is linked to users' cookies, and hence to their activities across the web. Worse, with many email clients, *third-party trackers receive the user's email address when the user views emails.* Further, when users click links in emails, regardless of the email client, we find additional leaks of the email address to trackers. These privacy breaches are our primary interest in this work.

We show that much of the time, leaks of email addresses to third parties are intentional on the part of commercial email senders. The resulting links between identities and web history profiles belie the claim of "anonymous" web tracking. The practice enables onboarding, or online marketing based on offline activity [9], as well as cross-device tracking, or linking between different devices of the same user [12]. And although email addresses are not always shared with third parties in plaintext—sometimes they are hashed—we argue that hashing does little to protect privacy in this context (Section 8).

Email tracking is possible because modern graphical email clients allow rendering a subset of HTML. JavaScript is invariably stripped, but embedded images and stylesheets are allowed. These are downloaded and rendered by the email client when the user views the email (unless they are proxied by the user's email server; of the providers we studied (Section 6.2), only Gmail and Yandex do so). Crucially, many email clients, and almost all web browsers, in the case of webmail, send third-party cookies with these requests, allowing linking to web profiles. The email address is leaked by being encoded as a parameter into these third-party URLs.

When links in emails pointing to the sender's website are clicked, the resulting leaks are outside the control of the email client or the email server. Even if the link doesn't contain any identifier, the web browser that opens the link will send the user's cookie with the request. The website can then link the cookie to the user's email address; this link may have been established when the user provided her email address to the sender via a web form. Finally, the sender can pass on the email address—and other personally identifiable information (PII), if available—to embedded third parties using methods such as redirects and referrer headers.

***Corresponding Author: Steven Englehardt:** Princeton University,
Jeffrey Han: Princeton University, E-mail:

***Corresponding Author: Arvind Narayanan:** Princeton University,

We now outline the methods we used, our findings, and our proposed defenses against email tracking.

1.1 Methods

Building on the OpenWPM web crawler [17], we created a tool to automatically search for mailing list subscription forms on websites and fill them in. It is challenging to scale such a tool due to numerous idiosyncrasies of websites (Section 3). Our crawler visited 15,700 sites and attempted to sign up for emails on each of these. The resulting corpus contains 12,618 emails from 902 distinct senders. The tool may be of independent interest for studying questions such as PII leakage from contact forms [38].

Next, we discuss how we detect instances of PII in network traffic (Section 4.1). This is a challenging problem because data might be encoded or hashed, possibly iteratively (e.g., double hashing or base-64 encoded and then hashed). In this study we focus exclusively on leaks of email addresses, but our techniques are agnostic to the type of PII. We examine leaks in network traffic resulting from a simulation of a user reading the corpus of emails collected as above (Section 4). We also simulated the user clicking on a sample of the links in the emails received, and looked for leaks in the resulting web traffic (Section 5).

We present a set of heuristics to classify such leakage as intentional or accidental (Section 4.1). Intentional leakage suggests a business relationship between the party sending the information and the party receiving it, whereas accidental leakage happens due to poor programming practices [23, 24].

Email providers (e.g., Gmail, employers) and email clients (e.g., Apple Mail, Thunderbird) may both employ measures to mitigate email tracking, such as proxying of images[1] or suppressing cookies. We built a tool that allows users and researchers to test the behavior of email providers and clients to assess the ability of email senders and third parties to track users. We use it ourselves to survey 16 email clients (Section 6.2).

1 Providers proxy resources by rewriting all remote resources in an email to point to a location on the provider's server. The provider requests the resource from the third-party server, rather than the user requesting it directly.

1.2 The state of email tracking

Email tracking is pervasive. We find that 85% of emails in our corpus contain embedded third-party content, and 70% contain resources categorized as trackers by popular tracking-protection lists. There are an average of 5.2 and a median of 2 third parties per email which embeds any third-party content, and nearly 900 third parties contacted at least once. But the top ones are familiar: Google-owned third parties (Doubleclick, Google APIs, etc.) are present in one-third of emails.

We simulate users viewing emails in a full-fledged email client (Section 4). We find that about 29% of emails leak the user's email address to at least one third party, and about 19% of senders sent at least one email that had such a leak. The majority of these leaks (62%) are intentional, based on our heuristics. Tracking protection is helpful, but not perfect: it reduces the number of email leaks by 87%. Interestingly, the top trackers that receive these leaked emails are different from the top web trackers. We present a case study of the most-common tracker, LiveIntent (Section 4.5).

We also simulate users clicking on links in emails, which causes a page to load in a full-fledged web browser (Section 5). We find that 11% of links contain embedded content requests that leak the email address to a third party, and at least 35% of senders include at least one such link in one email. The top third-party domains and organizations that receive these leaked email addresses are substantially similar to the list of top third parties overall.

1.3 Evaluating and improving defenses

We identify five possible defenses against email tracking: content proxying, HTML filtering, cookie blocking, referrer blocking, and request blocking. There are three possible ways to deploy defenses: by the mail server, the mail user agent, or the web user agent (i.e., the browser that handles links that are clicked on emails). We present a systematization of how each of these entities could deploy each of these defenses (Section 6.1).

The defenses that can be deployed by web browsers to protect against leaks of emails are nearly identical to defenses against web tracking in general. This is a mature area of research and there are numerous tools on the market based on filter lists. Based on our data analysis, we identify a list of 27,125 distinct URLs (from 133 domains) that receive leaked email addresses and are not blocked by prominent filter lists, presumably because these trackers are specific to emails (Section 7). We believe that these would make useful additions to existing filter lists. Except for this contribution, we focus our analysis of defenses on mail servers and mail user agents rather than web browsers.

Based on our analysis of 16 email servers and clients (Section 6.2) we find that a patchwork of defenses are employed, and no setup offers complete protection from the threats we identify. Perhaps the best option for privacy-conscious users today is to use webmail and install tracker-blocking extensions such as uBlock Origin or Ghostery.

We show that HTML filtering can be an effective defense. The idea is to rewrite email bodies to remove tracking elements. This can be done by either the mail server or the mail user agent. We prototype an element filtering tool based on existing tracking-protection lists and evaluate its effectiveness (Section 7).

2 Related work

Email secuity and privacy. The literature on email security and privacy has focused on authentication of emails and the privacy of email *contents*. For example, Durumeric et al. found that the long tail of SMTP servers largely fail to deploy encryption and authentication, leaving users vulnerable to downgrade attacks, which are widespread in the wild [15]. Holz et al. also found that email is poorly secured in transit, often due to configuration errors [21]. We study an orthogonal problem. Securing email in transit will not defend against email tracking, and vice versa.

Third-party web tracking. Email tracking is an outgrowth of third-party web tracking, which has grown tremendously in prevalence and complexity since the 1990s [13, 26, 28, 35]. Today Google is the most prominent tracker, through various third-party domains, and can track users across nearly 80% of sites [27]. Web tracking has expanded from simple HTTP cookies to include more persistent tracking techniques to "respawn" or re-instantiate HTTP cookies through Flash cookies [37], cache E-Tags, and HTML5 localStorage [10]. Overall, tracking is moving from stateful to stateless techniques: device fingerprinting attempts to identify users by a combination of the device's properties [16, 25]. Such techniques have advanced quickly [19, 30, 33], and are now widespread on the web [7, 8, 17, 32]. These techniques allow trackers to compile unique browsing histories, but they do not link histories to identity.

Compared to web tracking, email tracking does not use fingerprinting because (most) email clients prohibit JavaScript. On the other hand, email readily provides a unique, persistent, real-world identifier, namely the email address. Web tracking researchers have created a number of tools for detecting and measuring tracking and privacy, such as FPDetective [8], OpenWPM [17], and FourthParty [28]. We use OpenWPM for most of our measurements in this paper.

PII leakage. Leaks of PII of logged-in users from first-party websites to third parties are rampant; the early papers on this problem were by Krishnamurthy et al. [23, 24]. PII leaks enable trackers to potentially attach identities to browsing histories. More recent work includes detection of PII leakage to third parties in smartphone apps [34, 40], PII leakage in contact forms [38], PII leakage that enables cross-device tracking [12], and data leakage due to browser extensions [39].

The common problem faced by these authors (and by us) is that PII may be obfuscated. When the data collection is crowdsourced [34, 40] rather than automated, there is the further complication that the strings that constitute PII are not specified by the researcher and thus not known in advance. On the other hand, crowdsourced data collection allows obtaining numerous instances of each type of leak, which might make detection easier.

Various approaches are seen in prior work. Ren et al. employ heuristics for splitting fields in network traffic and detecting likely keys; they then apply machine learning to discriminate between PII and other fields [34]. Starov et al. apply differential testing, that is, varying the PII entered into the system and detecting the resulting changes in information flows [38]. This is challenging to apply in our context, because we observed frequent A/B testing in the commercial emails in our corpus, which makes it tricky to attribute observed changes to PII. This is an area for future work. Finally, our own approach is most similar to that of Brookman et al. [12] and Starov et al. [39] who test combinations of encodings and/or hashes.

3 Collecting a dataset of emails

We now describe how we assembled a large-scale corpus of mailing-list emails. We do not attempt to study a "typical" user's mailbox, since we have no empirical data from real users' mailboxes. Rather, our goal in assembling a large corpus is to study the overall landscape

High-level architecture of crawler.
Assemble a list of sites. For each site:

- *Find pages potentially containing forms.* For each page:
  - Find the best form on the page via *top-down form detection* and *bottom-up form detection.* If a form was found:
    - *Fill in the form*
    - *Fill in any secondary forms* if necessary
    - Once a form has been submitted, skip the rest of the pages and continue to next site

High-level architecture of server.
Receive and store email. For each email:

- *Check for and process confirmation links.*

Fig. 1. High-level architecture of the email collection system, with the individual modules italicized.

of third-party tracking of emails: identify as many trackers as possible (feeding into our enhancements to existing tracking-protection lists) and as many interesting behaviors as possible (such as different hashes and encodings of emails addresses).

To achieve scale, we use an automated approach. We created a web crawler based on the OpenWPM web privacy measurement tool [17] to search for and fill in forms that appear to be mailing-list subscriptions. The crawler has five modules, and the server that processes emails has two modules. They are both described at a high level in Fig. 1. We now describe each of the seven modules in turn.

Assemble a list of sites. Alexa maintains a public list of the top 1 million websites based on monthly traffic statistics, as well as rankings of the top 500 websites by category. We used the "Shopping" and "News" categories, since we found them more likely to contain newsletters. In addition, we visited the top 14,700 sites of the 1 million sites, for a total of 15,700 sites.

Detect and rank forms. When the crawler cannot locate a form on the landing page, it searches through all internal links (`<a>` tags) in the DOM until a page containing a suitable form is found. A ranked list of terms, shown in Table 1, is used to prioritize the links most likely to contain a mailing list. On each page, forms are detected using both a *top-down* and *bottom-up* procedure. The top-down procedure examines all fields contained in `<form>` elements. Forms which have a higher `z-index` and more input fields are given a higher rank, while forms which appear to be part of user account registration are given a lower rank. If no `<form>` elements are found, we attempt to discover forms contained in alternative containers (e.g., forms in `<div>` containers) using a bottom-up procedure. We start with each `<input>` element and recursively examine its parents until one with a submit button is found. For further details, see *Top-down form detection* and *Bottom-up form detection* in Appendix Section 10.1.

Fill in the form. Once a form is found, the crawler must fill out the input fields such that all inputs validate. The crawler fills all visible form fields, including: `<input>` tags, `<select>` tags (i.e., dropdown lists), and other submit `<button>` tags. Most websites use the general `text` type for all text inputs. We surveyed a number of top websites to determine common naming practices for input fields, and filled the fields with the data of the expected type. For example, name fields were filled with a generic first and last name. After submitting a form, we wait for a few seconds and re-run the procedure to fill follow-up fields, if required. For further details, see *Determining form field type* and *Handling two-part form submissions* in Appendix Section 10.1.

Receive and store email. We set up an SMTP server to receive emails. The server accepts any mail sent to an existing email address, and rejects it otherwise. It then parses the contents of the mail and logs metadata (such as the sender address, subject text, and recipient address) to a central database. All textual portions of the message contents are written to disk. We provide implementation details in Appendix Section 10.2.

Check for and process confirmation links. Our server will check the first email sent to each email address to determine if the mailing list requires additional user interaction to confirm the subscription. If the initial email's subject or *rendered* body text includes the keywords "confirm", "verify", "validate", or "activate", we extract potential confirmation links from the email. For HTML emails we collect links which match these keywords along with additional lower-priority keywords "subscribe" or "click". For plain-text emails we simply choose the longest link text. Emails with the past-tense keywords "confirmed", "subscribed", and "activated" in subject lines are skipped, as are links with the text "unsubscribe", "cancel", "deactivate", and "view". If any link is found, it is visited using OpenWPM.

Form submission measurement. Our crawler discovered and attempted to submit forms on 3,335 sites. We received at least one email from 1,242 (37%) of those sites. To understand the types of form submission failures, we ran a follow-up measurement in August 2017

Description	Keywords	Location
Email list registration	newsletter, weekly ad, subscribe, inbox, email, sale alert	link text
Generic registration	signup, sign up, sign me up, register, create, join	link text
Generic articles/posts	/article, news/, /2017	link URL
Selecting language/region	/us/, =us&, en-us	link URL
Blacklist	unsubscribe, mobile, phone	link text

Table 1. The web crawler chooses links to click based on keywords that appear in the link text or URL. The keywords were generated by iterating on an initial set of terms, optimizing for the success of mailing list sign-ups on the top sites. We created an initial set of search terms and manually observed the crawler interact with the top pages. Each time the crawler missed a mailing list sign-up form or failed to go to a page containing a sign-up form, we inspected the page and updated the set of keywords. This process was repeated until the crawler was successful on the sampled sites.

Submission classification		% of sampled sites
Total successful submissions		38%
→Mailing lists subscription	32%	
→User account registration	6%	
Failed: required a CAPTCHA		16%
Failed: unsupported form fields		25%
Unable to classify via screenshots		21%

Table 2. Submission success status of a sample of 252 of the 3,335 form submissions made during the sign-up crawl. The success and failure classification was determined through a manual review of screenshots taken before and after an attempted form submission.

where we took screenshots of the pages before and after the initial and follow-up form submissions. We manually examined a random sample of sites on which a form submission was attempted. We summarize the results in Table 2.

When filling forms, our crawler will interact with user account registration forms, mailing list sign-up forms, and contact forms. The successful submissions were mostly mailing list sign-ups and a small number of user account registrations, which are included as they can be tied to a mailing list. The failed submissions were mostly caused by forms other than mailing lists. In fact, more than 70% of the failures caused by a CAPTCHA or unsupported field were not mailing list form submissions. Overall, only 11% of the sampled mailing list interactions resulted in a CAPTCHA. Since our primary focus is mailing lists, we leave the evaluation of complex and CAPTCHA-protected forms to future work.

Email corpus. The assembled corpus contains a total of 12,618 HTML emails from 902 sites. We received an average of around 14 emails per site and a median of 5. A few sites had very active mailing lists, with 20 sites sending over 100 emails during the test period. We observe that we received no spam, which we confirmed both by manual inspection of a sample of emails as well as by finding an exact one-to-one correspondence between the 902 senders in our dataset and the unique email addresses that we generated. This ensures that the results represent the behavior of the sites where we registered, rather than spammers.

4 Privacy leaks when viewing emails

4.1 Measurement methodology

Simulating a webmail client. To measure web tracking in email bodies we render the emails using a simulated webmail client in an OpenWPM instance. Many webmail clients remove a subset of HTML tags from the email body to restrict the capabilities of rendered content. In particular, Javascript is exclusively removed, while iframe tags and CSS [6] have mixed support. We simulate a permissive webmail client, one which disables Javascript and removes the `Referer` header from all requests, but applies no other restrictions to the rendered content.

The email content is served on `localhost`, but is accessed through the domain localtest.me (which resolves to `localhost`) to avoid any special handling the browser may have for the local network. We configure OpenWPM to run 15 measurement instances in parallel. Each email is loaded twice in its own measurement instance: once with a fresh profile, and then again keeping the same browser profile after sleeping for 10 seconds. This is intended to allow remote content on the page to load both with and without browser state present. Indeed we observe some tracking images which redirect to new domains upon every subsequent reload of the same email.

Classifying third-party content. Many email clients load embedded content directly from remote servers (we further explore the properties of email clients in Section 6.2). Thus, remote content present in multiple emails can track users in the same way third-party content can track users across sites on the web. However, unlike the web there isn't always a clear distinction of which requests are "third-party" and which are "first-party". For example, all resources loaded by webmail clients are considered third-party by the browser. We consider any request to a domain[2] which is different than both the domain on which we signed up for the mailing list and the domain of the sender's email address to be a third-party request.

Detecting email leakage. Email addresses leak to remote servers through resource requests. Detecting these leaks is not as simple as searching for email addresses in requests, since the addresses may be hashed or encoded, sometimes iteratively. To detect such leakage we develop a methodology that, given a set of encodings and hashes, a plaintext email address, and a URL token, is able to determine if the token is a transformation of the email address. Starting with the plaintext email address we pre-compute a candidate set of tokens by applying all supported encodings and hashes iteratively, stopping once we reach three nested encodings or hashes. We then take the URL token and apply all supported decodings to the value, checking if the result is present in the candidate set. If not, we iteratively apply decodings until we reach a level of three nested decodings.

In a preliminary measurement we found no examples of a value that was encoded before being hashed. This is unsurprising, as hashed email addresses are used to sync data between parties and adding a transformation before the hash would prevent that use case. Thus, when analyzing the requests in this dataset, we restrict ourselves to at most three nested hashes for a set of 24 supported hashes, including `md5`, `sha1`, `sha256`. For encodings, we apply all possible combinations of 10 encodings, including `base64`, `urlencoding`, and `gzip`. The full list of supported hashes and encodings is given in Appendix 10.3.

Classifying email leakage. Email leaks may not be intentional. If an email address is included in the query string or path of a document URL it may automatically end up in the `Referer` header of subsequent requests from that document. Requests which result in a redirect also often add the referrer of the previous request to the query string of the new request. In many instances this happens irrespective of the presence of an email address in the original request. The situation is made more complex on the web since third-party Javascript can dynamically build URLs and trigger requests.

The reduced HTML support and lack of Javascript execution in email clients makes it possible to determine intentionality for most leaks. When an email is rendered, requests can result from three sources: from elements embedded in the original HTML, from within an embedded iframe (if supported by the client), or from a redirected request.

1. If a leak occurs in a `Referer` header it is **unintentional**. For webmail clients the `Referer` header (if enabled) will be the client itself. A mail sender can embed an iframe which loads a URL that includes the user's email address, with the explicit intention that the user's email leak to third parties via the `Referer` header. However, we chose not to include this possibility because email senders have multiple direct options for sharing information with third parties that do not rely on the sparsely supported iframe tag.
2. If a leak occurs in a request to a resource embedded directly in the HTML of the email body (and is not the result of a redirect) it is **intentional**. We can determine intentionality since any request resulting from an HTML document must have been constructed by the email sender. Note that this does not hold for web documents, since embedded Javascript can dynamically construct requests during the page visit.
3. If a request results from a redirect, the party responsible for the leak is the party whose request (i.e., the *triggering URL*) responded with a redirect to the new location (i.e., the *target URL*). We classify a leak as **intentional** if the leaked value is hashed between the triggering URL and the target URL, or if there are more encodings or hashes of the leaked value included in the target URL than in the triggering URL. If the target URL includes a full copy of the triggering URL (in any encoding) the leak is **unintentional**. All other cases are classified as **ambiguous**, such the case where a target URL includes only the query string of the triggering URL.

2 A domain is identified by its public suffix plus the component of the hostname immediately preceding its public suffix (PS+1).

Measuring blocked tags. Tracking protection tools which block resource requests offer users protection against the tracking embedded in emails. We evaluate the effectiveness of these tools by checking the requests in our dataset against two major blocklists, EasyList and EasyPrivacy [4]. These lists block advertisement and tracking related requests, and are bundled with several popular blocking extensions, including AdBlock Plus [1] and uBlock Origin [5]. We use the BlockList-Parser library [3] to determine if a request would have been blocked[3] by an extension utilizing these lists. We classify a request as blocked if it matches any of the following three conditions:

1. The request directly matches the filter list
2. The request is the result of a redirect and any request earlier in the redirect was blocked.
3. The request is loaded in an iframe and the iframe document request (or any resulting redirect) was blocked.

It is possible to do this classification in an offline fashion because of the lack of Javascript support in email clients. This removes the need to run measurements with one of the aforementioned extensions installed. In environments that support Javascript, content can be loaded dynamically and as the result of interactions between several scripts. In such an environment it is much more difficult to determine which requests would have been blocked by a single script appearing on the block list.

4.2 Email provides much of same tracking opportunities as the web

Remote resources embedded in email content can track users across emails. As we show in our survey of email clients (Section 6.2), many email clients allow remote resources to set persistent cookies and send those cookies with resource requests. In total, we find that 10,724 of the measured emails (85%) embed resources from at least one third party, with an average of 5 third parties per email. The distribution of embedded third parties is far from uniform; we find a median of two per email and a small number of emails embedding as many as 50 third parties (Figure 2).

3 We set the parser options as we would expect them to be set for a request occurring in a webmail client. For example, all requests are considered third-party requests.

Domain	% of Emails	% of Top 1M
doubleclick.net	22.2	47.5
mathtag.com	14.2	7.9
dotomi.com	12.7	3.5
adnxs.com	12.2	13.2
tapad.com	11.0	2.6
liadm.com	11.0	0.4
returnpath.net	11.0	<0.1
bidswitch.net	10.5	4,9
fonts.googleapis.com	10.2	39.4
list-manage.com	10.1	<0.1

Table 3. Top third-party domains by percentage of the 12,618 emails in the corpus. For comparison, we show the percentage of the top 1 million websites on which these third parties are present.

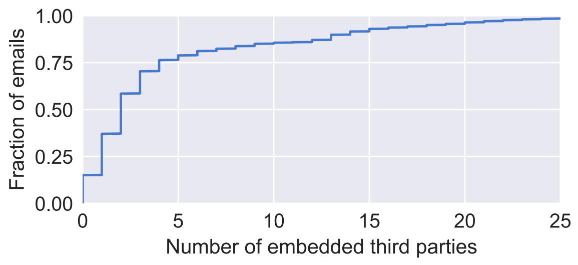

Fig. 2. CDF of third parties per email, aggregating data across the initial viewing and re-opening of an email. In addition, 1.4% of emails have between 25 and 53 third parties.

Table 3 shows the top third-party domains present in email. Many of these parties also have a large presence on the web [17], blurring the line between email and web tracking. On webmail clients, requests to these cross-context third parties will use the same cookies, allowing them to track both a user's web browsing and email habits. In total, the emails visited during our crawls embed resources from 879 third parties.

4.3 Leaks of email addresses to third parties are common

In addition to being able to track email habits, 99 third parties (11%) also gain access to a user's email address, whether in plaintext or hashed. In email clients which support cookies, these third parties will receive the email address alongside any cookies they've set on the user's device. Trackers which are also present on the web will thus be able to link this address with the user's browsing history profile.

Around 19% of the 902 senders leaked the user's email address to a third party in at least one email, and in total 29% of emails contain leaks to third parties. We find that a majority of these leaks, 62% of the 100,963 leaks to third parties, are intentional. These intentional leaks mostly occur through remote content embedded directly by the sender. Furthermore, 1% of leaks are classified as unintentional with the remainder considered ambiguous. While we do not attempt to determine how these identifiers are being used, plaintext and hashed emails can be used for persistent tracking, cross-device tracking, and syncing information between parties.

Leak	# of Senders	# of Recipients
MD5	**100 (11.1%)**	**38 (38.5%)**
SHA1	**64 (7.1%)**	**19 (19.2%)**
SHA256	**69 (7.6%)**	**13 (13.1%)**
Plaintext Domain	**55 (6.1%)**	**2 (2.0%)**
Plaintext Address	**77 (8.5%)**	**54 (54.5%)**
URL Encoded Address	**6 (0.6%)**	**8 (8.1%)**
SHA1 of MD5*	**1 (0.1%)**	**1 (1.0%)**
SHA256 of MD5*	**1 (0.1%)**	**1 (1.0%)**
MD5 of MD5*	**1 (0.1%)**	**1 (1.0%)**
SHA384	**1 (0.1%)**	**1 (1.0%)**

Table 4. Email address leakage to third parties by encoding. Percentages are given out of a total of 902 senders and 99 third-party leak recipients. All hashes are of the full email address. Email "domain" is the part of the address after the "@".
*These appear to be a misuse of LiveIntent's API (Section 4.5).

The leaked addresses are often hashed. Although we can detect email addresses hashed with 24 different functions and up to three nested layers, we only find `MD5`, `SHA1`, and `SHA256` in frequent use. Table 4 summarizes the number of senders and receivers of each encoding. The relatively low diversity of hashes and encodings suggests that these techniques are not being used to obfuscate the collection of email addresses. In fact, the query parameters which contain hashed emails sometimes identify the hash functions used in the parameter name (e.g., a string like `?md5=<md5 hash of email>` appearing in the HTTP request). The design of APIs like LiveIntent's, which first receives an email address and then syncs with a number of other parties (Section 4.5), suggests that these hashed address may be used to share or link data from multiple parties.

Recipient Organization	# of Senders
LiveIntent	**68 (7.5%)**
Acxiom	**46 (5.1%)**
Litmus Software	**28 (3.1%)**
Conversant Media	**26 (2.9%)**
Neustar	**24 (2.7%)**
apxlv.com	**18 (2.0%)**
54.211.147.17	**18 (2.0%)**
Trancos	**17 (1.9%)**
WPP	**17 (1.9%)**
54.82.61.160	**16 (1.8%)**

Table 5. Top organizations receiving email address leaks by number of the 902 total senders. A domain is used in place of an organization when it isn't clear which organization it belong to.

Table 5 identifies the top organizations[4] which receive leaked email addresses. This shows that email address collection from emails is largely consolidated to a few major players, which are mostly distinct from the popular web trackers. In fact, only one of the top 10 organizations, Neustar, is found in the top 20 third-party organizations on the top 1 million websites, as measured by Englehardt and Narayanan [17]. Also surprising is the prevalence of leaks to IP addresses, which accounts for eight of the top 20 domains receiving email addresses. This may be due to the relatively ephemeral nature of newsletter emails, which removes concerns of IP address churn over time.

4.4 Reopening emails brings in new third parties

Despite the lack of Javascript support, email views are dynamic. The email content itself is static, but any remote resources embedded in it may return different responses each time the email is viewed, and even redirect to different third parties. To examine the effects of this, we load every email first with a "clean" browser profile and then again without clearing the profile. Surprisingly, the average Jaccard similarity [36] between the sets of third parties loaded during the first and second views of the same email is only 60%.

The majority of emails—two-thirds—load fewer third parties when the email is reopened compared to the initial view. However, about 21% of emails load at

4 We map domains to organizations using the classification provided by Libert [27], adding several new email-specific organizations. When an organization could not be found, we use the PS+1.

Row	Request URL
0	http://inbox.washingtonexaminer.com/imp?[...]&e=<EMAIL>&p=0
1	http://p.liadm.com/imp?[...]&m=<MD5(address)>&sh=<SHA1(address)>&sh2=<SHA256(address)> &p=0&dom=<EMAIL_DOMAIN>
2	http://x.bidswitch.net/sync?ssp=liveintent&bidder_id=5298&licd=3357&x=EGF.M[...]
3	http://x.bidswitch.net/ul_cb/sync?ssp=liveintent&bidder_id=5298&licd=3357&x=EGF.M[...]
4	http://p.adsymptotic.com/d/px/?_pid=12688&_psign=d3e69[...]&bidswitch_ssp_id=liveintent&_redirect=[...]
5	http://p.adsymptotic.com/d/px/?_pid=12688&_psign=d3e69[...]&bidswit[...]&_redirect=[...]&_expected_cookie=[...]
6	http://x.bidswitch.net/sync?dsp_id=126&user_id=84f3[...]&ssp=liveintent
7	http://i.liadm.com/s/19751?bidder_id=5298&licd=3357&bidder_uuid=<UUID_1>
8	http://cm.g.doubleclick.net/pixel?google_nid=liveintent_dbm&google_cm&google_sc
9	http://cm.g.doubleclick.net/pixel?google_nid=liveintent_dbm&google_cm=&google_sc=&google_tc=
10	http://p.liadm.com/match_g?bidder_id=24314&bidder_uuid=<UUID_2>&google_cver=1
11	http://x.bidswitch.net/sync?ssp=liveintent&bidder_id=5298&licd=
12	http://pool.udsp.iponweb.net/sync?ssp=bidswitch&bidswitch_ssp_id=liveintent

Table 6. Redirect chain from a LiveIntent Email Tracking Pixel. URL query strings are truncated for clarity (using `[...]`).

least one resource when an email is reopened that wasn't present the first time. A small number of third parties are disproportionately responsible for this—they load different sets of additional third parties each time the email is opened (Table 14 in the Appendix).

The number of leaks between email loads stays relatively constant, with less than 50 emails leaking to new parties on the second load[5]. However, as the comparison of Table 14 with Table 5 shows, many of the top leak recipients are also responsible for redirecting to the highest number of new parties. Thus, reloading an email increases the number of potential recipients of a leak if the redirectors share data based on the email or email hash they receive.

4.5 Case study: LiveIntent

LiveIntent receives email addresses from the largest number of senders, 68 in total. In this section we analyze a sample of the request chains that result in leaks to LiveIntent. Table 6 shows an example redirect chain of a single pixel embedded in an email from the `washingtonexaminer.com` mailing list. The initial request (row 0) is to a subdomain of `washingtonexaminer.com`, and includes the user's plaintext email address in the `e=` query string parameter. The domain redirects to `liadm.com` (row 1), a LiveIntent domain, and includes the `MD5`, `SHA1`, and `SHA256` hashes of the email address in the parameters `m=`, `sh=`, and `sh2=`. The URL also includes the domain portion of the user's address.

In rows 2 - 12, the request redirects through several other domains and back to itself, exchanging what appear to be partner IDs and bidder IDs. In rows 7 and 10 LiveIntent receives a UUID from the domain in the previous request, which could allow it to exchange information with those trackers outside of the browser.

4.6 Request blockers help, but don't fix the problem

Privacy conscious users often deploy blocking extensions, such as uBlock Origin, Privacy Badger, or Ghostery, to block tracking requests. Since webmail clients are browser-based, these blocking extensions can also filter requests that occur while displaying email content[6]. We use our blocked tag detection methodology (Section 4.1) to determine which resources would have been blocked by the popular EasyList and EasyPrivacy blocklists. We then examine the remaining requests to determine how frequently email addresses continue to leak.

Overall, the blocklists cut the number of third parties receiving leaked email addresses from any sender nearly in half, from 99 to 51. Likewise, the number of senders which leak email addresses in at least one email is greatly reduced, from 19% to just 7%. However, as Table 7 shows, a significant number of leaks of both

5 We exclude leaks which occur to a different IP address on the second load. This occurs in 349 emails, but is less meaningful given the dynamic nature of IP address.

6 Thunderbird supports most of the popular Firefox extensions, and as such Thunderbird users can also deploy these defenses. See Table 12 for more details.

Encoding	# of Senders	# of Recipients
Plaintext Address	34 (3.7%)	34 (66.7%)
MD5	21 (2.3%)	12 (23.5%)
SHA1	14 (1.6%)	6 (11.8%)
URL Encoded Address	4 (0.4%)	4 (7.8%)
SHA256	4 (0.4%)	2 (3.9%)
SHA384	1 (0.1%)	1 (2.0%)

Table 7. Encodings used in leaks to third parties after filtering requests with EasyList and EasyPrivacy. Totals are given out of 902 email senders and 51 third-party leak recipients.

Recipient Domain	# of Senders
mediawallahscript.com	7
jetlore.com	4
scrippsnetworks.com	4
alocdn.com	3
richrelevance.com	3
ivitrack.com	2
intentiq.com	2
gatehousemedia.com	2
realtime.email	2
ziffimages.com	2

Table 8. The top third-party leak recipient domains after filtering requests with EasyList and EasyPrivacy. All recipients receive leaks from less than 1% of the 902 senders studied.

plaintext and email hashes still occur. In Table 8 we see that there are still several third-party domains which receive email address leaks, despite blocking. Several of these domains are known trackers which could be included in the blocklists. In addition, IP addresses and CDN domains are still recipients of leaked email addresses. Blocking on other URL features, such as the URL path, could help reduce leaks to these domains.

5 Privacy leaks when clicking links in emails

In Section 4 we explore the privacy impact of a user opening and rendering an email. In this section we explore the privacy impact of a user clicking links within an email. Once a user clicks a link in an email, the link is typically opened in a web browser. Unlike email clients, web browsers will typically support Javascript and advanced features of HTML, creating many potential avenues for privacy leaks. However, the only way an email address can propagate to a page visit is through the direct embedding of the address in a link contained in the original email body.

5.1 Measurement methodology

Sampling links from emails. To evaluate the privacy leaks which occur when links in emails are clicked, we generate a dataset from the HTML content of all emails and visit them individually in an instrumented browser. To extract the links from mail content, we parse all email bodies with `BeautifulSoup` [2] and extract the `src` property of all <a> tags. We sample up to 200 unique links per sender using the following sampling strategy. First, we bin links across all emails from a sender by the `PS+1` and `path` of the link. Next, we sample one link from each bin without replacement until there are no more links or we reach a limit of 200. This helps ensure that we have as diverse a set of landing pages as possible by stripping fragment and query string identifiers that may not influence the landing page.

Simulating link clicks. To simulate a user clicking a link, we visit each link in an OpenWPM instance using a fresh browser profile. The browser fully loads the page and sleeps for 10 seconds before closing. Unlike the email viewing simulation (Section 4), we enable both Javascript and `Referer` headers. This simulation replicates what happens when a link is clicked in a standalone email client; only the URL of the clicked link is passed to the browser for handling. In a webmail client, the initial request resulting from the click may also contain a cookie and a `Referer` header containing the email client's URL. We do not simulate these headers in our crawl.

Detecting email address leakage. To detect leakage of email addresses we use the procedure described in Section 4.1. Since the `Referer` header is enabled for these measurements, we consider a party to have received a leak if it is contained either in the URL or the `Referer` header of the resource request to that party. Email addresses can also be shared with the party through the `Cookie` header, request `POST` bodies, websocket connections, WebRTC connections, and so on. We consider these out of scope for this analysis.

5.2 Results

We found that about 11% of links contain requests that leak the email address to a third party. About 12% of all emails contain at least one such link, and among this subset, there are an average of 3.5 such links per email. The percentage of the 902 senders that leak the email address in at least one link in one email is higher: 35.5%. Finally, there were over 1,400 distinct third parties that

Recipient Organization	# of Senders
Google	247 (27.4%)
Facebook	160 (17.7%)
Twitter	94 (10.4%)
Adobe	81 (9.0%)
Microsoft	73 (8.1%)
Pinterest	72 (8.0%)
LiveIntent	69 (7.6%)
Akamai	69 (7.6%)
Acxiom	68 (7.5%)
AppNexus	61 (6.8%)

Table 9. The top leak recipient organizations based on a sample of simulated link clicks. All values are out of 902 total senders.

Recipient Domain	# of Senders
google-analytics.com	200 (22.2%)
doubleclick.net	196 (21.7%)
google.com	159 (17.6%)
facebook.com	154 (17.1%)
facebook.net	145 (16.1%)
fonts.googleapis.com	102 (11.3%)
googleadservices.com	96 (10.6%)
twitter.com	94 (10.4%)
googletagmanager.com	87 (9.6%)
gstatic.com	78 (8.6%)

Table 10. The top leak recipient domains based on a sample of simulated link clicks. All values are out of 902 senders.

received the email address in one or more of our simulated link clicks. We expect that all statistics in this paragraph, except the first, are slight underestimates due to our limit of 200 links per sender.

Table 9 shows the top organizations that receive leaked email addresses, and Table 10 shows the top domains. Over a quarter of senders leak the email address to Google in at least one link.

The most striking difference between these results and the corresponding results for viewing emails is that these lists look very similar to the list of top third party trackers [17], with the addition of a small number of organizations specific to email tracking. This motivates the privacy concern that identities could potentially be attached to third-party web tracking profiles.

6 Evaluation of defenses

6.1 Landscape of defenses

Defenses against tracking can be employed by several parties. We ignore mail senders and trackers themselves, since email tracking is a thriving commercial space and our evidence suggests that senders by and large cooperate with trackers to leak email addresses. We instead focus on parties who have an incentive to protect the recipient's privacy, namely the recipient's mail server, mail user agent, and the web browser.

The lines between these roles can be blurry, so we illustrate with two examples. Consider a user reading Yahoo mail via Firefox. The email server is Yahoo, the email client is Firefox together with Yahoo mail's client-side JavaScript, and the web browser is again Firefox. Or consider a user reading her university mail, via Gmail's IMAP feature, on her iPhone. For our purposes, both the university and Gmail count as email servers, since either of them is in a position to employ defenses. The email client is the Gmail iOS app, and the web browser is Safari.

Defense	Email server	Email client	Web browser
Content proxying	X		
HTML filtering	X	X	
Cookie blocking		X	X
Referrer blocking	X	X	X
Request blocking		X	X

Table 11. Applicability of each of the five possible defenses to each of the three contexts in which they may be deployed. An X indicates that the defense is applicable.

Table 11 summarizes the applicability of various defenses to the three roles. We discuss each in turn.

Content proxying. Email tracking is possible because of embedded content such as images and CSS (cascading style sheets). To prevent this, some email servers, notably Gmail, proxy embedded content. Thus, when the recipient views the email, the mail user agent does not make any requests to third parties.

This defense doesn't prevent the recipient email address being leaked to third parties, since it is leaked by being encoded in the URL. In fact, it *hinders* efforts by the mail client to prevent email address leakage (see request blocking below). However, it prevents third parties from learning the user's IP address, client device properties, and when the email was read (depending on how the proxy is configured). Most importantly, it prevents the third-party cookie from being sent, and thus prevents the third party from linking the user's email address to a tracking profile. In this way it is a complement to cookie blocking.

This defense can be deployed by the email server. Conceivably the email client might have its own server component through which embedded resources are proxied, but no email clients currently work this way, and further, it would introduce its own privacy vulnerabilities, so we ignore this possibility.

HTML filtering. HTML filtering refers to modifying the contents of HTML emails to mitigate tracking. It may be applied by the email server or the client, but it is more suitable to the server since the client can generally achieve the same effect in other ways, e.g., by request blocking or modifying the rendering engine. It is rarely applied today, and only in minimal ways. In Section 7 we prototype a comprehensive HTML filtering technique.

HTML filtering modifies the content of the email body, and thus might interfere with some email authentication methods, notably Domain Keys Identified Email (DKIM). However, since filtering is carried out by the recipient's mail server (Mail Transfer Agent) and not by intermediate mail relays, filtering can be done after the signature has been verified, and thus there is no impact on email authentication.

The following three techniques are applicable in one of two scenarios: when the email client requests embedded resources, or when the web browser handles clicks on links in emails.

Cookie blocking. Cookie blocking in the email client prevents third-party cookies from being sent when embedded content is requested. It is especially relevant in the webmail context, where the cookie allows third parties to link an email address to a web browsing profile. Even otherwise, blocking cookies is helpful since it makes it harder for third parties to compile a profile of the recipient's email viewing (they can always do this for the subset of emails where the email address is leaked).

Referrer blocking. If the email client sends the `Referer` header when loading embedded resources, it can allow several types of leaks. Depending on the implementation, the referrer may encode which client is being used and which specific email is being read. If the recipient forwarded an email to someone else and the email is being viewed in a different user's mailbox, it could leak this information. Worse, if the client supports iframes in emails, and the email address happens to be in the iframe URL, all requests to resources embedded in that iframe will accidentally leak the email address. For all these reasons, referrer blocking is a privacy-enhancing measure. There is little legitimate use for the referrer header in the context of email. While clients can certainly block the header (as can web browsers), servers can do this as well, by rewriting HTML to add the `rel="noreferrer"` attribute to links and inserting a Referrer Policy via the `meta` tag.

Request blocking. Request blocking is a powerful technique which is well known due to ad blockers and other browser privacy extensions. It relies on manually compiled *filter lists* containing thousands of regular expressions that define third-party content to be blocked. The most widely used ad-blocking list is EasyList, and the most widely used tracker-blocking list is EasyPrivacy. Filter list based blocking introduces false positives and false negatives [43], but the popularity of ad blocking suggests that many users find the usability trade-off to be acceptable. While request-blocking extensions are supported primarily by web browsers, some email clients also have support for them, notably Thunderbird.

6.2 Survey of email clients

We built an email privacy tester to discover which defenses are deployed by which popular email servers and clients.[7] Browser support for tracking protection has been extensively studied elsewhere [29], so we do not consider it here.

The email privacy tester allows the researcher to enter an email address and the name of an email client, and then sends an email to that address containing a tracking image and a link. The image and the link both have unique URLs. The researcher views the email in the specified email client, and then clicks on the link. The server records the following information: the email address, the email client, the IP address, timestamp, and headers sent for both the image and the link requests. The list of headers includes the cookie, referrer, and user agent.

We created accounts with a total of 9 email providers and tested them with a total of 16 email clients using various devices available in our lab. We analyzed the data recorded by the email privacy tester, and summarize the results in Table 12. We found that if defenses are deployed by email servers at all, they are only enabled for specific email clients (typically the default webmail client). Therefore we do not report on servers separately, but instead fold it into the analysis of clients. We also found that HTML filtering in a general form is not deployed, but only in the limited form of image and referrer blocking, so we report on that instead. We summarize our findings in Table 12.

7 https://emailtracking.openwpm.com/

Mail Client	Platform	Proxies Content	Blocks Images	Blocks Referrers	Blocks Cookies	Ext. Support
Gmail	Web	Yes	No*	L: Yes, I: Yes†	Yes†	Yes
Yahoo! Mail	Web	No	Yes	L: Yes, I: No	No	Yes
Outlook Web App	Web	No	Yes	No	No	Yes
Outlook.com	Web	No	No*	No	No	Yes
Yandex Mail	Web	Yes	No*	L: Yes, I: Yes†	Yes†	Yes
GMX	Web	No	No*	No	No	Yes
Zimbra	Web	No	Yes	No	No	Yes
163.com	Web	No	No*	No	No	Yes
Sina	Web	No	No	No	No	Yes
Apple Mail	iOS	No	No*	Yes	Yes	No
Gmail	iOS	Yes	No	Yes	Yes	No
Gmail	Android	Yes	No	Yes	Yes	No
Apple Mail	Desktop	No	No*	Yes	Yes	No
Windows Mail	Desktop	No	No*	Yes	No	No
Outlook 2016	Desktop	No	Yes	Yes	No	No
Thunderbird	Desktop	No	Yes	Yes	Optional (Default: No)	Yes

Table 12. A survey of the privacy impacting features of email clients. We explore whether the client proxies image requests, blocks images by default, blocks referrer headers from being sent (with image requests "I:" and with link clicks "L:"), blocks external resources from settings cookies, and whether or not the client supports request blocking extensions — either through the browser (for web clients) or directly (in the case of Thunderbird).
*Images are only blocked for messages considered spam.
† Blocking occurs as a result of proxied content.

7 Proposed defense

We argue that tracking protection should be at the center of a defensive strategy against email tracking. It can be employed either via HTML filtering on the server or via request blocking on the client. Tracking protection (and ad blocking) based on filter lists has proven to be effective and popular in web browsers, and its limitations manageable. The other defenses we examined all have serious drawbacks: for example, content proxying comes at a cost to the email server and makes email leaks *worse*, and cookie blocking is at best a partial solution.

We propose to improve tracking protection in two ways.

Server-side email content filtering. First, we prototype a server-side HTML filtering module. We use the existing, standard EasyList and EasyPrivacy filter lists. Our filtering script is written in Python using the BlockListParser library [3]. It scans for any HTML content (`text/html`) in email bodies, parses those contents, identifies embedded resources (images or CSS) whose URLs match one of the regular expressions in the filter lists, strips them out, and rewrites the HTML.

To test the effectiveness of HTML filtering, we ran our leak detection procedure on the filtered corpus of emails. We exclude one sender due to a measurement issue. We found that 11.0% of senders will leak email addresses to a third party in at least one email, and 11.5% of emails contain embedded resources which leak email to a third-party. Overall, 62 third parties received leaked email addresses, down from 99. As tracking-protection lists improve (see below), we can expect these numbers to decrease further. These numbers are very close to the corresponding numbers for request blocking (Section 4.6). The two techniques aren't identical: the one difference is that in static files, filtering is limited to the URLs present in the body of the HTML and will miss those that result from a redirect. However, this difference is small, and we conclude that HTML filtering is essentially as effective as request blocking.

Note that webmail users can already enjoy tracking protection, but server-side deployment will help all users, including those who use email clients that don't support request-blocking extensions.

Filling gaps in tracking-protection lists. As a second line of defense, we use our dataset to identify a list of 27,125 URLs representing 133 distinct parties which contain leaks of email addresses, but which aren't blocked by EasyList or EasyPrivacy. These include first parties in addition to third parties. We are able to identify first-party tracking URLs by observing groups of URLs of similar structure across different first-party domains. For example, 51 email senders leak the user's email address to a URL of the form `li.<public suffix + 1>/imp`, which appears to be part of LiveIntent's API (Section 4.5). We summarize the most common structures in the leaking URLs missed by tracking protection lists in Table 13.

URL Pattern	# of Senders
li.<PS+1>/imp	51 (5.7%)
partner.<PS+1>/	7 (0.7%)
stripe.<PS+1>/stripe/image	4 (0.4%)
p.<PS+1>/esp/open	4 (0.4%)
api.<PS+1>/layouts/section<N>	4 (0.4%)
<PS+1>/customer-service	3 (0.3%)
mi.<PS+1>/p/rp	3 (0.3%)
dmtk.<PS+1>/	3 (0.3%)
links.<PS+1>/e/open	3 (0.3%)
eads.<PS+1>/imp	3 (0.3%)

Table 13. The top URL patterns from URLs which leak email addresses and are missed by tracking protection lists (Section 4.6). The patterns are generated by stripping request URLs to hostname and path, replacing the public suffix plus one with `<PS+1>`, replacing integers with `<N>`, and stripping the last portion of the path if it ends with a file extension. The patterns are ranked by the number of senders which make at least one leaking request matching that pattern in any of the sender's emails. All values are given out of the total of 902 senders studied.

We suspect that the reason so many trackers are missed is that many of them are not active in the regular web tracking space. We have made the list of leaking URLs missed by tracking protection lists publicly available.[8] It should be straightforward to add regular expressions to filter lists based on these URLs; we suggest that filter list creators should regularly conduct scans of email corpora to identify new trackers.

8 Discussion and conclusion

Privacy risks of email tracking. Email security and privacy has not received much research attention despite its central importance in digital life. We showed that commercial emails contain a high density of third-party trackers. This is of concern not only because trackers can learn the recipient's IP address, when emails were opened, and so on, but also because these third parties are by and large the same ones that are involved in web tracking. This means that trackers can connect email addresses to browsing histories and profiles, which leads to further privacy breaches such as cross-device tracking and linking of online and offline activities. Indeed, email is an underappreciated avenue for straightforward cross-device tracking, since recipients tend to view emails on multiple devices.

The advice provided by many mail clients may mislead users into thinking the privacy risks associated with remote content are fairly limited. The remote content help pages of Gmail [20], Yahoo! Mail [42], and Thunderbird [31] all discuss the threat strictly in terms of the *email sender* learning information about the user, rather than a number of third parties.

Even network adversaries can benefit from the leaks in emails. The NSA is known to piggyback on advertising cookies for surveillance [18], and our work suggests one way in which a surveillance agency might attach identities to web activity records, in line with the findings of Englehardt et al. [18]. Indeed, nearly 91% of URLs containing leaks of emails are sent in plaintext.

Ineffectiveness of hashing. The putative justification for email address leaks in the online ad tech industry is that the address is hashed. However, hashing of PII, including emails, is not a meaningful privacy protection. This is folk knowledge in the security community, but bears repeating. Compared to hashing of passwords, there are several reasons why hashing of email addresses is far more easily reversible via variants of a dictionary attack. First, while (at least) some users attempt to maximize the entropy of passwords, most users aim to pick memorable emails, and hence the set of potential emails is effectively enumerable. Due to GPUs, trillions of hashes can be attempted at low cost. Second, unlike password hashing, salting is not applicable to email hashing since multiple third parties need to be able to independently derive the same hash from the email address.

Perhaps most importantly, if the adversary's goal is to retrieve records corresponding to a known email address or set of email addresses, then hashing is pointless—the adversary can simply hash the email addresses and then look them up. For example, if the adversary is a surveillance agency, as discussed above, and seeks to retrieve network logs corresponding to a given email address, this is trivially possible despite hashing.

Limitations. We mention several limitations of our work. First, despite the large number of heuristics that went into identifying and submitting forms, it is a fundamentally hard problem, and our crawler fails in many cases, including pages requiring complex mouse interactions, pages containing very poorly structured HTML, and CAPTCHA-protected form submission pages. Moreover, it is difficult to programmatically distinguish be-

8 https://gist.github.com/englehardt/6438c5d775ffd535b317d5c6ce3cde61

tween successful and failed form submissions. Looking at received network data is impractical, since responses could easily include text for both success and failure messages. On the other hand, looking only at changes in the rendered text on the webpage is more feasible, but would require handling many possible edge cases (e.g., page redirects, alerts, pop-up windows, iframes) and might still be too unreliable to use as a metric for success.

Second, our corpus of emails is not intended to be representative, and we are unable to draw conclusions about the extent of tracking in the typical user's mailbox.

Third, our simulation of a user viewing emails assumes a permissive user agent. We expect that this closely approximates a webmail setup with default browser settings (on browsers except Safari, which blocks third-party cookies by default), but we have not tested this assumption.

Future work. Finally, we mention several potential areas of future work.

Mailing list managers. It would be helpful to better understand the relationship between email senders and mailing list managers (such as Constant Contact). To what extent is email tracking driven by senders versus mailing list managers? When a sender sets up a marketing campaign with a mailing list manager, is the tracking disclosed to the sender?

PII leakage in registration forms. Researchers have previously found leakage of PII to third parties in contact forms on websites [38]. As far as we know, there has been no large-scale study of PII leakage in registration forms, where more sensitive information is often present (e.g. phone numbers, street addresses, and passwords). Recording and analyzing the third-party requests made *during* our crawls is an important area for further investigation.

Cookie syncing. It would be interesting to find out if cookie syncing occurs when viewing emails—a process in which different trackers exchange and link together their own IDs for the same user. Past work has shown that this happens among the vast majority of top third parties on the web [17], so we suspect that it occurs through email as well.

A/B testing. We notice some clear instances of A/B testing in our data, as might be expected in marketing campaigns. Specifically, we registered multiple email addresses on some sites at roughly the same time, and found several emails sent at nearly the same time (milliseconds apart) with different subject lines and email bodies advertising different products. We have not attempted to reverse-engineer or systematically analyze these differences, but it may be interesting to see if and how the received content changes in response to read receipts, click-through metrics, or other types of user interactions.

Differential testing. Despite testing for various encodings, hashes, and combinations, it is possible that we have missed some leaks of email addresses. We cannot hope to exhaustively test for all combinations of encodings and hashes. Instead, we propose differential testing: by registering multiple email addresses on the same site, we can look for parameters in URLs that are different for different email addresses, which are suggestive of transformed email addresses. The difficulty with this approach is that A/B testing, mentioned above, is a confound.

In summary, we hope that our work leads to greater awareness of the privacy risks of email tracking, spurs further research on the topic, and paves the way for deployment of robust defenses.

9 Acknowledgements

We would like to thank the anonymous reviewers, Aylin Caliskan, Paul-Olivier Dehaye, Joel Reardon, and Paul Van Oorschot for their helpful comments. We're also grateful to Günes Acar, Paul Ellenbogen, Marc Juarez, Harry Kalodner, Marcela Melara, and Laura Roberts for their assistance in compiling data for our email survey.

This work was supported by NSF Grant CNS 1526353, by a research grant from Mozilla, and by Amazon AWS Cloud Credits for Research.

References

[1] Adblock Plus - Surf the web without annoying ads! https://adblockplus.org/. Online; accessed 2017-09-05.

[2] BeautifulSoup. https://www.crummy.com/software/BeautifulSoup/. Online; accessed 2017-09-05.

[3] BlockListParser. https://github.com/shivamagarwal-iitb/BlockListParser. Online; accessed 2017-09-05.

[4] EasyList and EasyPrivacy. https://easylist.to/. Online; accessed 2017-09-05.

[5] uBlock Origin - An efficient blocker for Chromium and Firefox. Fast and lean. https://github.com/gorhill/uBlock/. Online; accessed 2017-09-05.

[6] CSS Support Guide for Email Clients. Campaign Source, https://www.campaignmonitor.com/css/ (Archive: https://www.webcitation.org/6rLLXBX0E), 2014.

[7] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of ACM CCS*, pages 674–689. ACM, 2014.

[8] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. Fpdetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1129–1140. ACM, 2013.

[9] Julia Angwin. Why online tracking is getting creepier. *ProPublica*, Jun 2014.

[10] Mika D Ayenson, Dietrich James Wambach, Ashkan Soltani, Nathan Good, and Chris Jay Hoofnagle. Flash cookies and privacy II: Now with html5 and etag respawning. 2011.

[11] Bananatag. Email Tracking for Gmail, Outlook and other clients. https://bananatag.com/email-tracking/. Online; accessed 2017-09-04.

[12] Justin Brookman, Phoebe Rouge, Aaron Alva Alva, and Christina Yeung. Cross-device tracking: Measurement and disclosures. In *Proceedings of the Privacy Enhancing Technologies Symposium*, 2017.

[13] Ceren Budak, Sharad Goel, Justin Rao, and Georgios Zervas. Understanding emerging threats to online advertising. In *Proceedings of the ACM Conference on Economics and Computation*, 2016.

[14] ContactMonkey. Email Tracking for Outlook and Gmail. https://www.contactmonkey.com/email-tracking. Online; accessed 2017-09-04.

[15] Zakir Durumeric, David Adrian, Ariana Mirian, James Kasten, Elie Bursztein, Nicolas Lidzborski, Kurt Thomas, Vijay Eranti, Michael Bailey, and J Alex Halderman. Neither snow nor rain nor mitm...: An empirical analysis of email delivery security. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pages 27–39. ACM, 2015.

[16] Peter Eckersley. How unique is your web browser? In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 1–18. Springer, 2010.

[17] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *ACM Conference on Computer and Communications Security*, 2016.

[18] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th Conference on World Wide Web*, 2015.

[19] David Fifield and Serge Egelman. Fingerprinting web users through font metrics. In *International Conference on Financial Cryptography and Data Security*, 2015.

[20] Gmail Help. Choose whether to show images. https://support.google.com/mail/answer/145919. Online; accessed 2017-09-06.

[21] Ralph Holz, Johanna Amann, Olivier Mehani, Mohamed Ali Kâafar, and Matthias Wachs. TLS in the wild: An internet-wide analysis of tls-based protocols for electronic communication. In *23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[22] HubSpot. Start Email Tracking Today. https://www.hubspot.com/products/sales/email-tracking. Online; accessed 2017-09-04.

[23] Balachander Krishnamurthy, Konstantin Naryshkin, and Craig Wills. Privacy leakage vs. protection measures: the growing disconnect. In *Proceedings of the Web*, 2011.

[24] Balachander Krishnamurthy and Craig E Wills. On the leakage of personally identifiable information via online social networks. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 7–12. ACM, 2009.

[25] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *37th IEEE Symposium on Security and Privacy*, 2016.

[26] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *25th USENIX Security Symposium*, 2016.

[27] Timothy Libert. Exposing the invisible web: An analysis of third-party http requests on 1 million websites. *International Journal of Communication*, 9:18, 2015.

[28] Jonathan R Mayer and John C Mitchell. Third-party web tracking: Policy and technology. In *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012.

[29] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar Weippl. Block me if you can: A large-scale study of tracker-blocking tools. In *Proceedings of the 2nd IEEE European Symposium on Security and Privacy (IEEE EuroS&P)*, 2017.

[30] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. *W2SP*, 2012.

[31] Mozilla Support. Remote Content in Messages. https://support.mozilla.org/en-US/kb/remote-content-in-messages. Online; accessed 2017-09-04.

[32] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Security and privacy (SP), 2013 IEEE symposium on*, pages 541–555. IEEE, 2013.

[33] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. The leaking battery A privacy analysis of the HTML5 Battery Status API. Technical report, 2015.

[34] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374. ACM, 2016.

[35] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 12–12. USENIX Association, 2012.

[36] scikit-learn. Jaccard Similarity Score. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.jaccard_similarity_score.html. Online; accessed 2017-09-05.

[37] Ashkan Soltani, Shannon Canty, Quentin Mayo, Lauren Thomas, and Chris Jay Hoofnagle. Flash cookies and privacy. In *AAAI spring symposium: intelligent information privacy management*, volume 2010, pages 158–163, 2010.

[38] Oleksii Starov, Phillipa Gill, and Nick Nikiforakis. Are you sure you want to contact us? quantifying the leakage of pii

via website contact forms. *Proceedings on Privacy Enhancing Technologies*, 2016(1):20–33, 2016.

[39] Oleksii Starov and Nick Nikiforakis. Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1481–1490, 2017.

[40] Narseo Vallina-Rodriguez, Christian Kreibich, Mark Allman, and Vern Paxson. Lumen: Fine-grained visibility and control of mobile traffic in user-space. 2017.

[41] W3C. 4.10 Forms - HTML5. https://www.w3.org/TR/html5/forms.html. Online; accessed 2017-09-07.

[42] Yahoo Help. Block images in your incoming Yahoo Mail emails. https://help.yahoo.com/kb/SLN5043.html. Online; accessed 2017-09-06.

[43] Zhonghao Yu, Sam Macbeth, Konark Modi, and Josep M Pujol. Tracking the trackers. In *Proceedings of the 25th International Conference on World Wide Web*, pages 121–132. International World Wide Web Conferences Steering Committee, 2016.

10 Appendix

10.1 Form discovery and filling methodology

Choosing pages on which to search for forms. The crawler searches through all links (<a> tags) on the landing page to find pages that are most likely to contain a mailing list form. It does this by matching the link text and URL against a ranked list of terms, which are shown in Table 1. As an initial step, we filter out invisible links and links to external sites. We check that the link text does not contain words in our blacklist, which aims to avoid unsubscribe pages and phone-based registration. If we have found any links that match, the crawler clicks on the one with the highest rank, then runs the form-finding procedure on the new page and any newly opened pop-up windows. If no forms are found, it goes back and repeats this process for the remaining links. The reason for clicking on generic article links is that we have come across several news sites with newsletter forms only within article pages. We also make sure to select the English language or US/English locale when available, since our keywords are in English.

Top-down form detection. For each page the crawler visits, it first searches through the HTML DOM for any potential email registration forms. When sites use the standard `<form>` element, it can simply iterate through each form's input fields (`<input>` tags) and see if any text fields ask for an email address (by matching on input type and keywords). If so, it marks the form as a candidate, and then chooses the best candidate using the following criteria (in order):

1. Always return the topmost form. Any form stacked on top of other elements is probably a modal or dialog, and we find that the most common use of these components is to promote a site's mailing lists. We rely on the z-index CSS property, which specifies the stacking order of an element in relation to others (as a relative, arbitrary integer). Note that most DOM elements take the default z-index value of `auto`, inheriting the actual value from its parent; thus, the crawler recursively checks a form's parent elements until it finds a non-`auto` value, or reaches the root of the DOM tree. To break ties, it also searches for the literal strings "modal" or "dialog" within the form's HTML, since we find that such components are usually descriptively named.
2. Rank login forms lower. This is the other class of forms that often asks for an email address, so the crawler explicitly checks for the strings "login", "log in", and "sign in" within a form's HTML to avoid these when other candidates are present.
3. Prefer forms with more input fields. This is mainly helpful for identifying the correct follow-up form: if we submit our email address in the footer of a page, the same footer might be present on the page we get redirected to. In this scenario, the form we want to pick is the longer one.

Additionally, registration forms are sometimes found inside of inline frames (`<iframe>` tag), which are effectively separate HTML pages embedded in the main page. If necessary, we iterate through each frame and apply the same procedure to locate registration forms within them.

Bottom-up form detection. A growing number of sites place logical forms inside of generic container elements (e.g., `<div>` or `<span>` tags), without using any `<form>` tags. Therefore if top-down form detection fails, we take a bottom-up approach: the crawler first iterates through all the `<input>` elements on the page to check if any email address fields exist at all, then recursively examines their parents to find the first container that also contains a submit button. This container is usually the smallest logical form unit that includes all of the relevant input fields.

Determining form field type. Once a form is discovered, we need to determine which fields are contained in the form and fill each field with valid data. We skip any invisible elements, since a real user would not be expected to fill them. Some fields can be identified by their type attribute alone—for example, `tel` for phone numbers and `email` for email addresses—but these specific types were introduced in the relatively recent HTML5 standard [41], and most websites still use the general `text` type for all text inputs. In our survey of the top sites, we found that contextual hints are scattered across many tag attributes, with the most frequent being `name`, `class`, `id`, `placeholder`, `value`, `for`, and `title`. In addition, tags that contain HTML bodies (such as `<button>` tags) often contain hints in the `innerHTML`.

Handling two-part form submissions After submitting a form, we are sometimes prompted to fill out another longer form before the registration is accepted. This second form might appear on the same page (i.e., using JavaScript), or on a separate page either through a redirect or as a pop-up window. We take a simplistic approach: the crawler waits a few seconds, then applies the same form-finding procedure first on any pop-up windows and then on the original window. This approach may have the effect of submitting the same form twice, but we argue that this does not produce any adverse results—duplicate form submissions are a plausible user interaction that web services should be expected to handle gracefully.

10.2 Mail server implementation

The mail server receives emails using SubEtha SMTP, a library offering a simple low-level API to handle incoming mail. The server accepts any mail sent to (`RCPT TO`) an existing email address, and rejects it otherwise. The mail contents (`DATA`) are parsed in MIME format using the JavaMail API, and the raw message contents are written to disk. MIME messages consist of a set of headers and a content body, with the required `Content-Type` header indicating the format of the content; notably, a *multipart* content body contains additional MIME message subparts, enabling messages to be arranged in a tree structure. To save disk space, we recursively scan multipart MIME messages for subparts with content types that are non-text (`text/*`), such as attached images or other data, and discard them before storing the messages since we do not examine any non-textual content.

10.3 Supported hash functions and encodings for leak detection

Supported hashes and checksums: md2, md4, md5, sha, sha1, sha256, sha224, sha384, sha3-224, sha3-256, sha3-384, sha3-512, murmurhash2 (signed and unsigned), murmurhash3 32-bit, murmurhash3 64-bit, murmurhash3 128-bit, ripemd160, whirlpool, blake2b, blake2s, crc32, adler32

Supported encodings: base16, base32, base58, base64, urlencoding, deflate, gzip, zlib, entity, yenc

10.4 Top parties redirecting to new third parties on email reload

Redirecting Party	Organization	Avg add'l parties	#S	#E
pippio.com	Acxiom	5.7	7	32
liadm.com*	LiveIntent	3.7	68	1097
rlcdn.com	Acxiom	1.7	11	551
imiclk.com	MediaMath	1.3	2	4
mathtag.com	MediaMath	1.1	11	382
alcmpn.com	ALC†	0.8	6	132
emltrk.com	Litmus	0.7	41	638
acxiom-online.com	Acxiom	0.4	2	33
dyneml.com	PowerInbox	0.1	3	13
adnxs.com	AppNexus	0.1	19	277

Table 14. Top parties by average number of new third-party resources in a redirect chain when an email is reloaded. The number of senders (# S) out of 902 total and the number of emails (#E) out of 12,618 total on which this occurs is given for each redirecting party. We exclude redirecting parties that only exhibit this behavior in emails from a single sender. In total, there are 12 parties which exhibit this type of redirect behavior.
* Includes statistics for chains which redirect to http://p.liadm.com/imp in the first redirect. We observe a common pattern of URLs of the form li.firstparty.com redirecting first to this endpoint which then redirects to a number of other third parties.
† American List Counsel